Web Designing

Part-3

educlash.com
Just another Way To Learn

6

FORMS

Unit Structure

1. OBJECTIVE

After going through this chapter you will be able to:

- Learn how to place form in a web page
- Explain how to put different form elements like text field, password, file upload field, submit and reset button, checkbox, radio button, select field
- Learn how to validate a form using script
- Learn how to use event handler with forms
- Learn how to pass form data to another page

2. CREATING FORMS

Forms are the most popular way to make web pages interactive. Like forms on paper, a form on a web page allows the user to enter requested information and submit it for processing. (Fortunately, forms on a web page are processed much faster.)

For creating form in a web page we use <FORM> tag. <FORM> tag is a container tag which can have many other form elements in it like text fields, radio buttons, checkboxes, etc.

3. THE <FORM> TAG

<FORM > indicates the beginning of a form. All other form tags go inside <FORM>. In its simplest use, <FORM>can be used without any attributes.

Attributes that can be used with form tags are as follows:

Name	Description	Example
URL	ACTION gives the URL of the program which will process this form. For example, the CGI program "MyCGI" is located at../cgi-bin/mycgi.pl (you can go directly to that URL). This form uses "MyCGI":	<FORM ACTION="../cgi-bin/mycgi.pl"> favorite color: <INPUT NAME=COLOR><BR> <INPUT TYPE=SUBMIT> </FORM> When you click submit, your browser sends the form data to the CGI indicated in ACTION.

METHOD	METHOD = GET | POST METHOD specifies the method of transferring the form data to the web server. METHOD can be either GET or POST. Each method has its advantages and disadvantages. GET sends the data as part of the URL. POST is the preferred method for sending lengthy form data. When a form is submitted POST the user does not see the form data that was sent.	<FORM METHOD=GET ACTION="../cgi-bin/mycgi.pl"> town: <INPUT NAME="town"><BR> <INPUT TYPE=SUBMIT> </FORM> The value entered in the "town" field is tacked on to the end of the CGI's URL like this: ../cgi-bin/mycgi.pl?town=West+Rochester <FORM METHOD=POST ACTION="../cgi-bin/mycgi.pl">

NAME	NAME = "text string"
	
	NAME gives a name to the form. This is most useful in scripting, where you frequently need to refer to the form in order to refer to the element *within* the form.
		<SCRIPT>
		<!--
		function Circle_calc_ii()
		{
		var CircleRadius = document.MyCircleForm.Circle_radius.value;
		if (CircleRadius >= 0)
		{
		document.MyCircleForm.Circle_circumference.value = 2 * Math.PI * CircleRadius ;
		document.MyCircleForm.Circle_area.value = Math.PI * Math.pow(CircleRadius, 2) ;
		}
		else
		{
		document.MyCircleForm.Circle_circumference.value = "";

		document.MyCircleForm.Ci
		
		rcle_area.value = "";
		        }
		    }
		// -->
		</SCRIPT>
		
		<FORM
		
		NAME="MyCircleForm">
		<TABLE BORDER
		CELLPADDING=3>
		<TR>
		<TD><NOBR>radius:
		<INPUT
		NAME="Circle_radius"
		SIZE=4></NOBR></TD>
		<TD><INPUT
		TYPE=BUTTON
		OnClick="Circle_calc_ii(thi
		s.form);"
		VALUE="calculate"></TD
		>
		<TD ALIGN=RIGHT
		BGCOLOR="#AACCFF">
		    <NOBR>circumference:
		<INPUT
		NAME="Circle_circumfere
		nce"
		SIZE=9></NOBR><BR>
		    <NOBR>area: <INPUT
		NAME="Circle_area"
		SIZE=9></NOBR></TD>
		</TR>
		
		</TABLE>
		
		</FORM>

ENCTYPE	ENCTYPE = "multipart/form-data" | "application/x-www-form-urlencoded" | "text/plain" ENCTYPE determines how the form data is encoded. Whenever data is transmitted from one place	

	to another, there needs to be an agreed upon means of representing that data. Music is translated into written music notation, English is written using letters and punctuation. Similarly, there needs to be an agreed on way of presenting the form data so it's clear that, for example, there is a field called "email" and its value is "abc@docs.com". In most cases you will not need to use this attribute at all. The default value is "application/x-www-form-urlencoded", which is sufficient for almost any kind of form data. The one exception is if you want to do file uploads. In that case you should use "multipart/form-data".	

TARGET	TARGET = "_blank" | "_parent" | "_self" | "_top" | frame name
	
	TARGET indicates which frame in a set of frames to send the results to, and works just like <A TARGET="...">. This attribute can be used so that the form is always visible even as the form results are displayed and redisplayed.
		<FORM
		TARGET="TargetFrame"
		ACTION="../cgi-bin/mycgi.pl">

onSubmit	onSubmit = "script command(s)"
	
	onSubmit is a scripting event that occurs when the user attempts to submit the form. onSubmit can be used to do some error checking on the form data, and to cancel the submit if an error is found.
	
	Note that in order to cancel the submit event, the onSubmit should be in the form onSubmit="return expression". "return" indicates that value of
		This <FORM> tag calls a Javascript function to check the form data:
		
		<FORM
		ACTION="../cgi-bin/mycgi.pl"
		NAME="testform"
		onSubmit="return TestDataCheck()">

	the expression should be returned to submit routine. If expression evaluates to false, the submit routine is cancelled; if it is true, the submit routine goes forward.	
onReset	onReset = "script command(s)" onReset runs a script when the user resets the form. If onReset returns false, the reset is cancelled.	

Let’s check your progress:

1. Name given to form tag is used by___________
   a. form elements b. script c. form d.table

6.3 NAMED INPUT FIELDS

Whenever we are placing form on a web page for example registrations form. We require various form elements in that form. Each form element should be given a unique name. This unique name will be helpful to us when we are referring these elements in a scripting code.

```
<SCRIPT>
<!--
function Circle_calc_ii()
  {
  var CircleRadius = document.MyCircleForm.Circle_radius.value;
  if (CircleRadius >= 0)
    {
    document.MyCircleForm.Circle_circumference.value =  2 * Math.PI
* CircleRadius ;
    document.MyCircleForm.Circle_area.value =  Math.PI *
Math.pow(CircleRadius, 2) ;
    }
  else
    {
    document.MyCircleForm.Circle_circumference.value =  "";
    document.MyCircleForm.Circle_area.value =  "";
    }
  }
// -->
```

```
</SCRIPT>
<FORM NAME="MyCircleForm">
<TABLE BORDER CELLPADDING=3>
<TR>
<TD>radius: <INPUT type= text NAME="Circle_radius" SIZE=4></TD>
<TD><INPUT TYPE=BUTTON OnClick="Circle_calc_ii(this.form);"
VALUE="calculate"></TD>
<TD ALIGN=RIGHT BGCOLOR="#AACCFF">
   circumference: <INPUT type= text NAME="Circle_circumference"
SIZE=9><BR>
   area: <INPUT type= text NAME="Circle_area" SIZE=9></TD>
</TR>
</TABLE>
</FORM>
```

Code Explanation:

In above code we have three text fields each one has got distinct name i.e. Circle_radius, Circle_cimcumference, Circle_area. When we want to refer to these fields in our scripting code, we write as follows:

document.MyCircleForm.**Circle_radius**.value

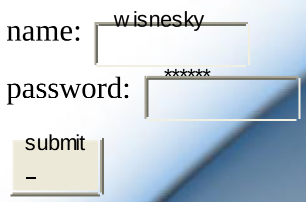
document.MyCircleForm.**Circle_circumference**.value = 2 * Math.PI * CircleRadius ;

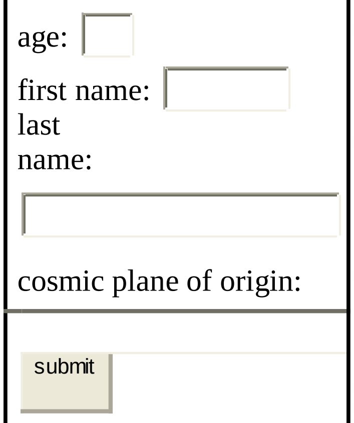
document.MyCircleForm.**Circle_area**.value = Math.PI * Math.pow(CircleRadius, 2);

6.4 THE <INPUT> TAG

<INPUT> creates the data entry fields on an HTML form. Attributes that can be used with <INPUT> tag are as follows:

Name	**Description**	**Example**
TYPE	TYPE = TEXT | CHECKBOX | RADIO | PASSWORD | HIDDEN | SUBMIT | RESET |BUTTON | FILE | IMAGE TYPE establishes what type of data entry field this is.	
NAME	NAME assigns a name to the input field, and is required in most circumstances. In forms which use scripts, the name of the input field is sent to the script.	<FORM ACTION="../cgi-bin/mycgi.pl" onSubmit="return (this.email.value != '')" > email: <INPUT

	The input object is in the elements collection of the form object, and can be referred to by its name using dot notation.	NAME="email"> <P><INPUT TYPE=SUBMIT VALUE="submit"> </FORM> In this example, we use the this.form.email to refer to the email input field.
VALUE	VALUE sets the value for the input field. VALUE sets the default values for text and password fields, sets the button text in submit, reset and plain buttons, sets the values of the choices in radio buttons, sets the permanent values of hidden fields, and has no effect on file, and image fields.	<FORM ACTION="../cgi-bin/mycgi.pl"> name: <INPUT TYPE=TEXT NAME="realname" VALUE="wisnesky"><BR> password: <INPUT TYPE=PASSWORD NAME="realname" VALUE="pacman"> <P><INPUT TYPE=SUBMIT VALUE="submit"> </FORM> **Output:**  name: wisnesky password: ****** submit
SIZE	SIZE = integer SIZE sets how wide a text or password field should be. It has no effect on any other type of field. SIZE does not set the maximum length of what can be typed in	<FORM ACTION="../cgi-bin/mycgi.pl"> age:<INPUT TYPE=TEXT NAME="age" SIZE=2 ><BR> first name: <INPUT TYPE=TEXT NAME="first" SIZE=10><BR> last name: <INPUT TYPE=TEXT

		NAME="last" SIZE=30><BR> cosmic plane of origin:<BR> <INPUT TYPE=TEXT NAME="plane" SIZE=70> <P><INPUT TYPE=SUBMIT VALUE="submit"> </FORM> **Output:**  age: first name: last name: cosmic plane of origin:
MAXLENGTH	MAXLENGTH = integer MAXLENGTH sets the maximum number of characters for text or password fields	account ID: <INPUT TYPE=TEXT NAME="accountID" MAXLENGTH=4><BR > password: <INPUT TYPE=PASSWORD NAME="password" MAXLENGTH=8> In account ID field we can enter maximum 4 characters and in password field maximum 8 characters.
CHECKED	CHECKED indicates that a radio button or checkbox should be on when the form first loads.	<FORM ACTION="../cgi-bin/mycgi.pl"> <INPUT TYPE=CHECKBOX NAME="maillist" CHECKED>Yes! Put

		me on the list!
 What color would you like?
 <INPUT TYPE=RADIO NAME="color" VALUE="green" >Green
 <INPUT TYPE=RADIO NAME="color" VALUE="red" >Red
 <INPUT TYPE=RADIO NAME="color" VALUE="blue" CHECKED >Blue
 <INPUT TYPE=RADIO NAME="color" VALUE="brown" >Brown <P><INPUT TYPE=SUBMIT VALUE="submit"> </FORM> **Output:** ☑ Yes! Put me on the list! What color would you like? ○ Green ○ Red ◉ Blue ○ Brown submit
BORDER	BORDER = integer BORDER is used for image submit buttons. BORDER indicates if there should be a visible border around the image. BORDER only has an effect in Netscape. MSIE does not put any visible border around image submits.	

	When you use the image type of input, you can use many of the same attributes as with <IMG >, including: SRC = "image URL" HEIGHT, WIDTH, ALT, HSPACE = integer, VSPACE = integer, ALIGN = LEFT | RIGHT | TOP | TEXTTOP | MIDDLE | ABSMIDDLE | CENTER | BOTTOM | ABSBOTTOM | BASELINE	
DISABLED, READONLY	READONLY and DISABLED both remove the functionality of the input field, but to different degrees. READONLY locks the field: the user cannot change the value. DISABLED does same thing but takes it further: user cannot use the field in any way, not to highlight the text for copying, not to select the checkbox, not to submit the form. In fact, a disabled field is not even sent if the form is submitted.	<INPUT NAME="realname" VALUE="Hi There" READONLY> <INPUT NAME="realname" VALUE="Hi There" DISABLED>

ACCESSKEY	ACCESSKEY = "text string"
	ACCESSKEY specifies a shortcut key to go directly to the input field. The key is pressed along with the ALT key. For button style fields, using the key is like pressing the button.
		<INPUT TYPE=SUBMIT
		ACCESSKEY="g"
		VALUE="Go!">

TABINDEX	TABINDEX = integer Normally, when the user tabs from field to field in a form (in a browser that allows tabbing, not all browsers do) the order is the order the fields appear in the HTML code. However, sometimes you want the tab order to flow a little differently. In that case, you	<FORM ACTION="../cgi-bin/mycgi.pl" METHOD=POST> <TABLE BORDER CELLPADDING=3 CELLSPACING=5 BGCOLOR="#FFFFCC">

	can number the fields using TABINDEX. The tabs then flow in order from lowest TABINDEX to highest.	<TR> <TD>name: <INPUT NAME="realname" TABINDEX=1></TD> <TD ROWSPAN=3>comments<BR> <TEXTAREA COLS=25 ROWS=5 TABINDEX=4> </TEXTAREA></TD></TR> <TR> <TD>email: <INPUT NAME="email" TABINDEX=2></TD></TR> <TR> <TD>department: <SELECT NAME="dep" TABINDEX=3> <OPTION VALUE="">... <OPTION VALUE="mkt">Marketing <OPTION VALUE="fin">Finance <OPTION VALUE="dev">Development <OPTION VALUE="prd"> Production</SELECT></TD></TR> </TABLE> </FORM>

Let's check your progress:

2. How form elements can be placed using <INPUT> tag? a. 8 b.7 c.9 d. 10

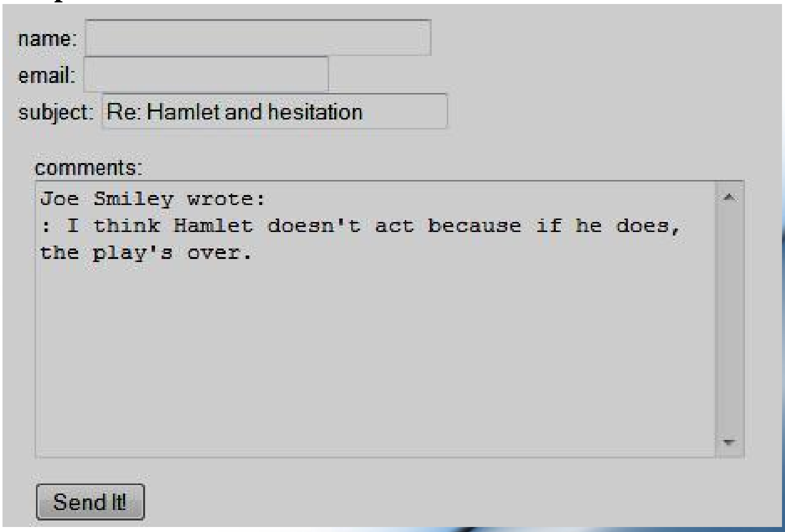
3. A ____________field is not even sent if the form is submitted
   a. disabled b. enabled c. readonly d. blank

6.5 HIDDEN

HIDDEN indicates that the field is invisible and the user never interacts with it. The field is still sent to the script, and scripts can also use the hidden field. HIDDEN is commonly used as output of a script which creates a new form for more input. For example, a web site which facilitates online discussions may use a hidden field to keep track of which message is being responded to:

```
<H2>Your Reply</H2>
<FORM METHOD=POST ACTION="../cgi-bin/mycgi.pl">
<INPUT TYPE=HIDDEN NAME="postingID" value="98765">
name:    <INPUT NAME="realname" SIZE=30><BR>email:   <INPUT
NAME="email"><BR>
subject: <INPUT NAME="subject" VALUE="Re: Hamlet and hesitation"
SIZE=30>
<P>comments:<BR>
<TEXTAREA NAME="comments" COLS=50 ROWS=10
WRAP=VIRTUAL>
Joe Smiley wrote:
: I think Hamlet doesn't act because if he does, the play's over.
</TEXTAREA>
<P><INPUT TYPE=SUBMIT VALUE="Send It!">
</FORM>
```

Output:



name:	
email:	
subject:	Re: Hamlet and hesitation

comments:

```
Joe Smiley wrote:
: I think Hamlet doesn't act because if he does,
the play's over.
```

Code Explanation:
In the above code PostID field is hidden so in the output you will not be able to see that field. But definitely the value of PostID is passed to the next page. So values which you want to pass to next page but without displaying to user then you can hidden field.

Let’s check your progress:

4. Which fields are not visible to user on web page?
   a. readonly b. hidden c. disabled d.visual

6.6 TEXT

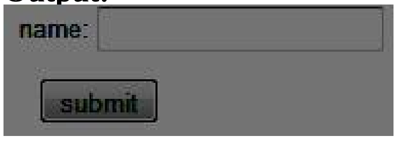
TEXT creates a text entry field (the most popular type of data entry field):

```
<FORM ACTION="../cgi-bin/mycgi.pl">
name: <INPUT TYPE=TEXT NAME="realname">
<P><INPUT TYPE=SUBMIT VALUE="submit">
</FORM>
```

OR

```
<FORM ACTION="../cgi-bin/mycgi.pl">
<!-- Note absence of TYPE attribute -->
name: <INPUT NAME="realname">
<P><INPUT TYPE=SUBMIT VALUE="submit">
</FORM>
```

Output:



The behavior of TEXT fields can be modified using these attributes:
VALUE: set an initial value for the field
SIZE: how wide the field should be
MAXLENGTH: the maximum number of characters the user can enter

6.7 TEXT AREA

<TEXTAREA> indicates a form field where the user can enter large amounts of text (i.e. multiline input). In most respects, <TEXTAREA> works like an <INPUT> field. It can have a name, a default value, script events such as onChange, and is sent to a scripting code as a name/value pair. One main difference is that <TEXTAREA> is a container tag.

<TEXTAREA> takes following attributes:

Name	Description	Example
NAME	NAME = "text string" NAME sets the name of the field for use in scripting. This attribute works just like <INPUT NAME="...">	

COLS & ROWS	COLS indicate how many characters (not pixels) wide the text area should be. ROWS indicate how many rows should be in the text area. Both attributes are required in the <TEXTAREA> tag. These attributes do not set any limit on how much can be typed in, just how much of the textarea is visible	<TEXTAREA NAME="few" COLS=10 ROWS=2></TEXTAREA> **Output:** <TEXTAREA NAME="some" COLS=50 ROWS=5></TEXTAREA> **Output:**
DISABLED READONLY	DISABLED and READONLY work exactly like the corresponding attributes for <INPUT>. Please see <INPUT DISABLED>and <INPUT READONLY>.	
TABINDEX	TABINDEX = integer TABINDEX is supported by MSIE 4.x and higher and Netscape 6. Normally, when the user tabs from field to field in a form (in a browser that allows tabbing, not all browsers do) the tab order is the order the fields appear in the HTML code. However, sometimes you want the tab order to flow a little differently. In that case, you can number the fields using TABINDEX. The tabs then flow in order from lowest TABINDEX to highest.	<TABLE BORDER CELLPADDING=3 CELLSPACING=5 BGCOLOR="#ffffcc"> <TR><TD>name: <INPUT NAME="realname" TABINDEX=1></TD> <TD ROWSPAN=3>comments<BR> <TEXTAREA COLS=25 ROWS=5 TABINDEX=4></TEXTAREA></TD></TR> <TR> <TD>email: <INPUT NAME="email" TABINDEX=2></TD></TR> <TR> <TD>department: <SELECT NAME="dep" TABINDEX=3>

		<OPTION VALUE="">... <OPTION VALUE="mkt">Marketing <OPTION VALUE="fin">Finance <OPTION VALUE="dev">Development <OPTION VALUE="prd"> Production</SELECT></TD> </TR> </TABLE>

Let's check your progress:

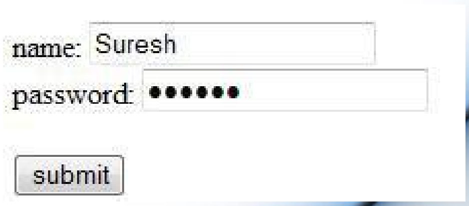
5. Which field allows user to enter multiple line data?
   a.text b. password c. textarea d.hidden

6.8 PASSWORD

PASSWORD indicates that the field is for typing in a password. PASSWORD works just like a TEXT type field, with the difference that whatever is typed is not displayed on the screen (in case someone is watching over your shoulder or you have to leave the work station). Instead of showing what you typed in, the browser displays a series of asterisks (*), bullets (·), or something to show that you are typing, but not what you are typing. So, for example, this code:

```
<FORM ACTION="../cgi-bin/mycgi.pl" METHOD=POST> name:
<INPUT TYPE=TEXT NAME="realname"><BR> password:
<INPUT TYPE=PASSWORD NAME="mypassword">
<P><INPUT TYPE=SUBMIT VALUE="submit">
</FORM>
```

Output:

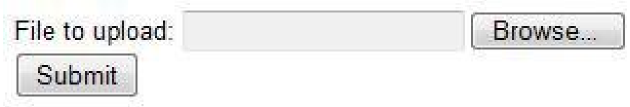

name: Suresh
password: ●●●●●●

Note that PASSWORD fields are not sent encrypted, they are sent in the same manner as all the other elements on the form: in the clear text. Note also that when you use PASSWORD you should also set the form METHOD to POST.

6.9 FILE UPLOAD

FILE is used for doing file uploads in a form. File uploads are a relatively new and still not well-standardized type of form input, but they show great promise once the bugs are ironed out. File uploads allow you to send an entire file from your computer to the web server as part of your form input.

```
<FORM METHOD=POST ENCTYPE="multipart/form-data"
ACTION="../cgi-bin/mycgi.pl">
File to upload: <INPUT TYPE=FILE NAME="upfile"><BR>
<INPUT TYPE=SUBMIT VALUE="Submit">
</FORM>
```



Configuring a form for file uploads requires setting two attributes in the <FORM> tags in addition to using <INPUT TYPE=FILE>: POST and "multipart/form-data" (as in the example above). When the data is sent, the original file name (including the full path) of the file as it was on your computer is sent to the web server. The Scripting code, however, is free to save the file as anything it wants -- or to not save it at all.

An often expressed wish with file uploads is to have a way of suggesting the file type being uploaded. Netscape, for example, when it gives you the file upload dialog box, inexplicably assumes you want to upload an HTML file. Unfortunately, there is no way to suggest a file type.

Let’s check your progress:

6. What should be the value of ENCTYPE if file upload filed is used in web page?
   a. multipart/form-data   b. multipart   c. text/form-data
   d. form-data

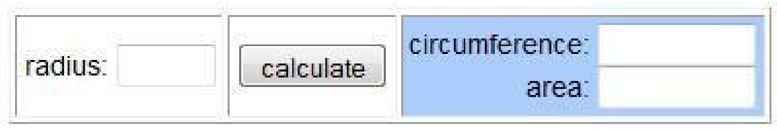
6.10 BUTTON

BUTTON defines a button which causes a script to run. Use the onClick attribute to give the script command(s). BUTTON is used only with scripting. Browsers that don't understand scripts don't understand this type of input and usually render it as a text input field.

```
<FORM>
<TABLE BORDER CELLPADDING=3>
<TR>
```

```
   <TD><NOBR>radius: <INPUT NAME="Circle_radius"
SIZE=4></NOBR></TD>
   <TD><INPUT TYPE=BUTTON
OnClick="Circle_calc(this.form);" VALUE="calculate"></TD>
   <TD ALIGN=RIGHT BGCOLOR="#AACCFF">
   <NOBR>circumference: <INPUT
NAME="Circle_circumference" SIZE=9></NOBR><BR>
   <NOBR>area: <INPUT NAME="Circle_area"
SIZE=9></NOBR></TD>
   </TR>
</TABLE>
</FORM>
```

Output:

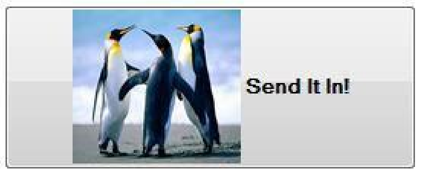


<BUTTON>:
Another way to render button on a web page is using <BUTTON> tag. <BUTTON> creates a button. Unlike <INPUT>, <BUTTON> is a container which allows you to put regular HTML contents in the button, including text and pictures. Unfortunately, <BUTTON> does not degrade well, and so at this time it's best to stick with <INPUT>. <BUTTON> tag has following attributes:

TYPE: what type of button is this	DISABLED: disable this button
onClick: script to run when the user clicks here	ACCESSKEY: shortcut key for this button
NAME: name of this button element	TABINDEX: tab order
VALUE: the value sent with the form	

```
<BUTTON TYPE=SUBMIT><IMG SRC="Penguins.jpg" HEIGHT=97
WIDTH=105 ALT="Starflower"
ALIGN="ABSMIDDLE"><STRONG>Send It
In!</STRONG></BUTTON>
```

Output:



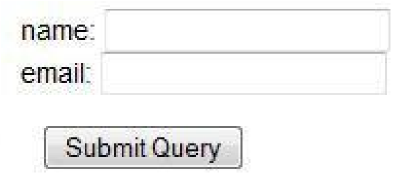
By default, <BUTTON> creates a plain button, much like <INPUT TYPE=BUTTON>. With the TYPE attribute, <BUTTON> can also create submit and reset buttons. The HTML code put between <BUTTON> and </BUTTON> is not the value sent with the form. The value of the button determined by the <INPUT VALUE="..."> attribute.

SUBMIT

SUBMIT creates the "Submit" button which sends the form in to the CGI. In its simplest form, you can use SUBMIT and no other attributes for the <INPUT ...> tag:

```
<FORM ACTION="../cgi-bin/mycgi.pl">
name: <INPUT NAME="realname"><BR>
email: <INPUT NAME="email"><P>
<INPUT TYPE=SUBMIT>
</FORM>
```

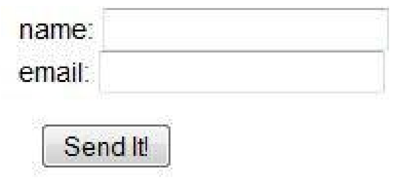
Output:



You can customize the text used for the button using the VALUE attribute:

```
<FORM ACTION="../cgi-bin/mycgi.pl">
name: <INPUT NAME="realname"><BR>
email: <INPUT NAME="email"><P>
<INPUT TYPE=SUBMIT VALUE="Send It!">
</FORM>
```

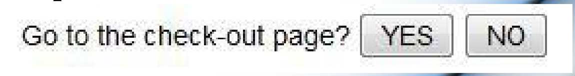
Output:



You may sometimes find that you want to have more than one submit button on a form. If you give each button the same name, but different values, the browser will indicate which submit button was pressed:

```
<FORM ACTION="../cgi-bin/mycgi.pl">Go to the check-out
page?
<INPUT TYPE=SUBMIT NAME="checkout" VALUE="YES">
<INPUT TYPE=SUBMIT NAME="checkout" VALUE="NO">
</FORM>
```

Output:



Let's check your progress:

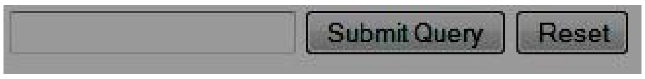
7. Can we place more than one submit button in an html form?
   a. yes b. no

RESET

RESET resets the form so that it is the way it was before anything was typed in:

```
<FORM ACTION="../cgi-bin/mycgi.pl">
<INPUT TYPE=TEXT>
<INPUT TYPE=SUBMIT>
<INPUT TYPE=RESET>
</FORM>
```

Output:



If you do choose to use have a reset button in your form, consider adding a check if the user actually wants to reset. You can do this by adding an onReset event handler to the <FORM> tag:

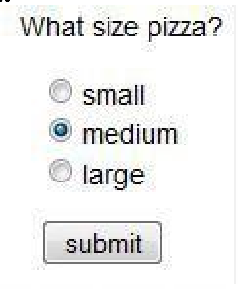
```
<FORM ACTION="../cgi-bin/mycgi.pl"
  onReset="return confirm('Do you really want to reset the form?')">
<INPUT TYPE=TEXT NAME="query">
<INPUT TYPE=SUBMIT>
<INPUT TYPE=RESET>
</FORM>
```

6.11 RADIO

RADIO is used to create a series of choices of which only one can be selected. The term "radio button" comes from the buttons for the radio in an automobile, where selecting one radio station automatically de-selects all the others. HTML radio buttons are created by using several <INPUT TYPE=RADIO> buttons, all with the same name, but with different values. For example, this series of buttons allows you to choose one size for a pizza:

```
<FORM ACTION="../cgi-bin/mycgi.pl">
What size pizza?<P>
<INPUT TYPE=RADIO NAME="pizzasize" VALUE="S"
>small<BR>
<INPUT TYPE=RADIO NAME="pizzasize" VALUE="M"
CHECKED >medium<BR>
<INPUT TYPE=RADIO NAME="pizzasize" VALUE="L"
>large<P>
<INPUT TYPE=SUBMIT VALUE="submit">
</FORM>
```

Output:



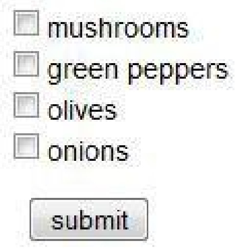
If no CHECKED attribute is used, different browsers have different ways of displaying the initial state of a series of radio buttons. Netscape and MSIE have none of the buttons selected. Mosaic selects the first button.

6.12 CHECKBOX

CHECKBOX creates a checkbox which can be either on or off:

```
<FORM ACTION="../cgi-bin/mycgi.pl">
<INPUT TYPE=CHECKBOX NAME="mushrooms"
>mushrooms<BR>
<INPUT TYPE=CHECKBOX NAME="greenpeppers">green
peppers<BR>
<INPUT TYPE=CHECKBOX NAME="olives"      >olives<BR>
<INPUT TYPE=CHECKBOX NAME="onions"      >onions<P>
<INPUT TYPE=SUBMIT VALUE="submit">
</FORM>
```

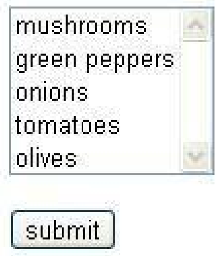
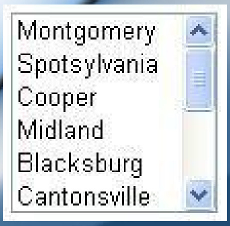
Output:



By default, the checkbox is initially off. If you want the checkbox initially on, use the CHECKED attribute. Checkbox CHECKBOXs are only sent to the scripting code if they are on; if they are off, no name/value pair is sent (try out the form above to see).

6.13 SELECT

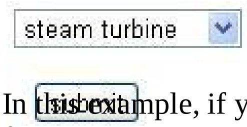
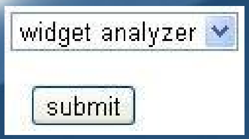
<SELECT> sets up a list of choices from which a user can select one or many. <SELECT> tag takes following attributes:

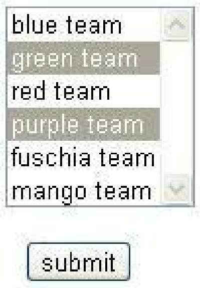
Name	Description	Example
NAME	NAME names the select field for use with scripting. NAME works just like <INPUT NAME="...">	
MULTIPLE	MULTIPLE designates that more than one option in the list can be selected. When creating a multiple list it is almost always a good idea to also use the SIZE attribute.	<SELECT NAME="toppings" MULTIPLE SIZE=5> <OPTION VALUE="mushrooms">mushrooms <OPTION VALUE="greenpeppers">green peppers <OPTION VALUE="onions">onions <OPTION VALUE="tomatoes">tomatoes <OPTION VALUE="olives">olives </SELECT>  mushrooms green peppers onions tomatoes olives submit
SIZE	SIZE = integer SIZE indicates how many rows of the list should be displayed. The default is one.	For example, the following code creates a select list with 6 rows <SELECT NAME="county" SIZE=6>  Montgomery Spotsylvania Cooper Midland Blacksburg Cantonsville

DISABLED	DISABLED works exactly the same as the corresponding attribute for <INPUT>.	
TABINDEX	TABINDEX = integer Normally, when the user tabs from field to field in a form (in a browser that allows tabbing, not all browsers do) the tab order is the order the fields appear in the HTML code. However, sometimes you want the tab order to flow a little differently. In that case, you can number the fields using TABINDEX. The tabs then flow in order from lowest TABINDEX to highest.	<TABLE BORDER CELLPADDING=3 CELLSPACING=5 BGCOLOR="#FFFFCC"> <TR> <TD>name: <INPUT NAME="realname" TABINDEX=1></TD> <TD ROWSPAN=3>comments
 <TEXTAREA COLS=25 ROWS=5 TABINDEX=4></TEXTAREA></TD></TR> <TR> <TD>email: <INPUT NAME="email" TABINDEX=2></TD></TR> <TR> <TD>department: <SELECT NAME="dep" TABINDEX=3> <OPTION VALUE="">... <OPTION VALUE="mkt">Marketing <OPTION VALUE="fin">Finance <OPTION VALUE="dev">Development <OPTION VALUE="prd">Production</SELECT></TD></TR> </TABLE>

6.14 OPTION

<OPTION ...> is used along with <SELECT> to create select lists. <OPTION> indicates the start of a new option in the list. <OPTION> can be used without any attributes, but you usually need the VALUE attribute, which indicates what is sent to the server. The text which follows <OPTION> is what is displayed in the browser. <OPTION> takes following attributes:

Name	**Description**	**Example**
VALUE	VALUE indicates the value that is sent to the server if that option is chosen. The value of VALUE is not seen by the user.	<SELECT NAME="partnumber"> <OPTION VALUE="7382">steam turbine <OPTION VALUE="2928">resistor array <OPTION VALUE="3993">widget analyzer <OPTION VALUE="9398">fiber identifier </SELECT> **Output:**  In this example, if you selected the first option, "steam turbine", then the name/value pair partnumber=7382 is sent to the scripting code.
SELECTED	SELECTED indicates that the option should be selected by default.	In this example, the third item ("widget analyzer") is the default item: <SELECT NAME="partnumber"> <OPTION VALUE="7382" >steam turbine <OPTION VALUE="2928" >resistor array <OPTION VALUE="3993" SELECTED >widget analyzer <OPTION VALUE="9398" >fiber identifier </SELECT> **Output:** 

		SELECTED can also be used in multiple select lists. In this example, the "green team" and the "purple team" are the default selected items: <SELECT NAME="teams" MULTIPLE SIZE=6> <OPTION VALUE="b" >blue team <OPTION VALUE="g" SELECTED >green team <OPTION VALUE="r" >red team <OPTION VALUE="p" SELECTED >purple team <OPTION VALUE="f" >fuschia team <OPTION VALUE="m" >mango team </SELECT> **Output:**  blue team green team red team purple team fuschia team mango team submit

Let's check your progress:

8. If you want to a drop down box with five cities name in it. How many <OPTION> tags are required?
   a. 2 b.3 c.5 d.1

6.15 FORMS AND SCRIPTING

Source code:

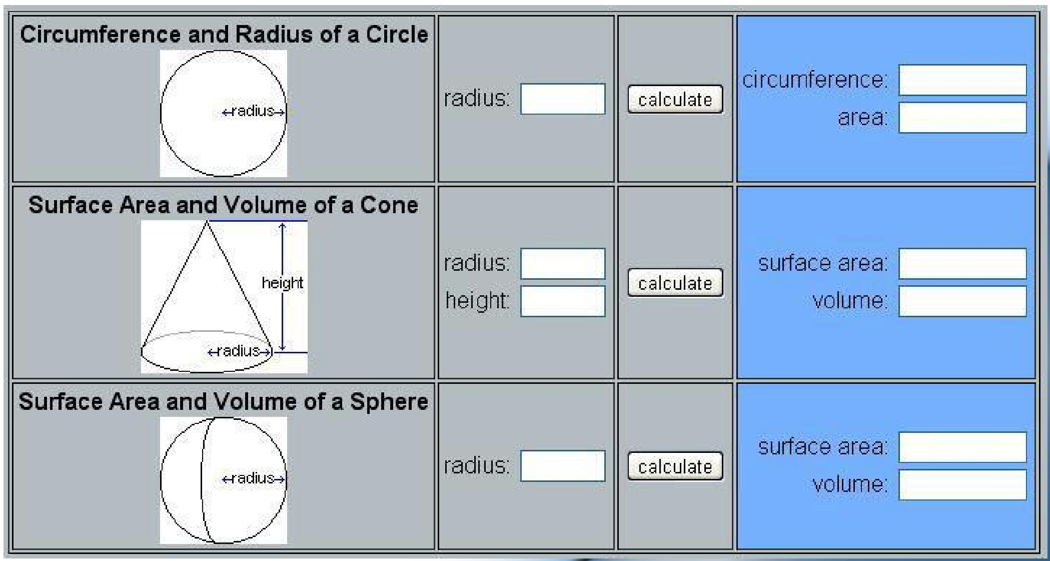
```
<SCRIPT TYPE="text/javascript">
<!--
function Circle_calc(GeoForm)
{
var CircleRadius = GeoForm.Circle_radius.value;
```

```
if (CircleRadius >= 0)
  {
  GeoForm.Circle_circumference.value =  2 * Math.PI * CircleRadius ;
  GeoForm.Circle_area.value =  Math.PI * Math.pow(CircleRadius, 2) ;
  }
else
  {
  GeoForm.Circle_circumference.value =  "";
  GeoForm.Circle_area.value =  "";
  }
}
function Cone_calc(GeoForm)
{
var ConeRadius = GeoForm.Cone_radius.value;
var ConeHeight = GeoForm.Cone_height.value;
if ((ConeRadius >= 0) && (ConeHeight >= 0))
  {
  GeoForm.Cone_surfacearea.value = (Math.PI * Math.pow(ConeRadius,
2)) +
(Math.PI * ConeRadius * Math.sqrt(Math.pow(ConeRadius, 2) +
Math.pow(ConeHeight, 2)));
  GeoForm.Cone_volume.value = (1/3) * Math.PI *
Math.pow(ConeRadius,2) *
ConeHeight;
  }
else
  {
  GeoForm.Cone_surfacearea.value = "";
  GeoForm.Cone_volume.value = "";
  }
}
function Sphere_calc(GeoForm)
{
var SphereRadius = GeoForm.Sphere_radius.value;
if (SphereRadius >= 0)
  {
  GeoForm.Sphere_surfacearea.value = 4 * Math.PI *
Math.pow(SphereRadius, 2);
  GeoForm.Sphere_volume.value = (4/3) * Math.PI *
Math.pow(SphereRadius, 3);
```

```
  }
else
  {
  GeoForm.Sphere_surfacearea.value = "";
  GeoForm.Sphere_volume.value = "";
  }
}
// -->
</SCRIPT>
<FORM>
<TABLE BORDER CELLPADDING=3>
<!-- circumference and radius of a circle -->
<TR>
<TH>Circumference and Radius of a Circle<BR><IMG
SRC="../graphics/circle.gif"
HEIGHT=88 WIDTH=88 ALT="picture of a circle"></TH>
<TD><NOBR>radius: <INPUT NAME="Circle_radius"
SIZE=4></NOBR></TD>
<TD><INPUT TYPE=BUTTON OnClick="Circle_calc(this.form);"
VALUE="calculate"></TD>
<TD ALIGN=RIGHT BGCOLOR="#AACCFF">
  <NOBR>circumference: <INPUT NAME="Circle_circumference"
SIZE=9></NOBR><BR>
  <NOBR>area: <INPUT NAME="Circle_area"
SIZE=9></NOBR></TD>
</TR>
<!-- Surface area and volume of a cone -->
<TR>
<TH>Surface Area and Volume of a Cone<BR><IMG
SRC="../graphics/cone.gif" HEIGHT=106 WIDTH=115 ALT="picture of
a cone"></TH>
<TD ALIGN=RIGHT><NOBR>radius: <INPUT NAME="Cone_radius"
SIZE=4></NOBR><BR>
  <NOBR>height: <INPUT NAME="Cone_height"
SIZE=4></NOBR></TD>
<TD><INPUT TYPE=BUTTON OnClick="Cone_calc(this.form);"
VALUE="calculate"></TD>
<TD ALIGN=RIGHT BGCOLOR="#AACCFF">
  <NOBR>surface area: <INPUT NAME="Cone_surfacearea"
SIZE=9></NOBR><BR>
```

```
   <NOBR>volume: <INPUT NAME="Cone_volume"
SIZE=9></NOBR></TD>
</TR>
<!-- Surface Area and Volume of a Sphere -->
<TR>
<TH>Surface Area and Volume of a Sphere<BR><IMG
SRC="../graphics/sphere.gif" HEIGHT=88 WIDTH=88 ALT="picture of
a sphere"></TH>
<TD><NOBR>radius: <INPUT NAME="Sphere_radius"
SIZE=4></NOBR></TD>
<TD><INPUT TYPE=BUTTON OnClick="Sphere_calc(this.form);"
VALUE="calculate"></TD>
<TD ALIGN=RIGHT BGCOLOR="#AACCFF"><NOBR>surface area:
<INPUT NAME="Sphere_surfacearea" SIZE=9></NOBR><BR>
   <NOBR>volume: <INPUT NAME="Sphere_volume"
SIZE=9></NOBR></TD>
</TR>
</TABLE>
</FORM>
```

Output:



Explanation:
In the above code three functions are defined in <SCRIPT> tag namely Circle_calc(GeoForm), Cone_calc(GeoForm), Sphere_calc(GeoForm).

Circle_calc(GeoForm) function calculates circumference and area of a circle based on the value of radius entered by user(value of radius is stored using the code: var CircleRadius = GeoForm.Circle_radius.value;).

This value of radius is used in formula to calculate circumference and area and displayed in respective text fields.

Similarly Cone_calc(GeoForm), Sphere_calc(GeoForm) are defined for cone and sphere.

These functions are called in the HTML code when user clicks on calculate button for circle, cone and sphere (i.e. onclick event of calculate button).

6.16 LABELING INPUT FILES

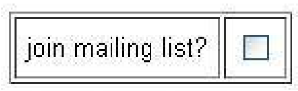
<LABEL ...>, an HTML 4.0 element supported by MSIE and Netscape 6, defines a set of text that is associated with a particular form element.

<LABEL> tag takes one attribute i.e. FOR:
FOR = "text string"

If the form element that should be associated with the text in <LABEL> can't be contained within <LABEL>, such as when the form element in another table cell, use FOR. The value of FOR is the ID of the form element. Note that it is not the name of the field (as given with the NAME attribute) it is the ID as given with the ID attribute. So, for example, this code associates the text "join mailing list?" with the checkbox field that has the ID "joinlist":

```
<TABLE BORDER=1 CELLPADDING=5>
<TR> <TD><LABEL FOR="joinlist">join mailing
list?</LABEL></TD>
    <TD><INPUT TYPE=CHECKBOX NAME="joinlist"
ID="joinlist"></TD>
    </TR>
</TABLE>
```

Output:



6.17 GROUPING RELATED FIELDS

<FIELDSET>:

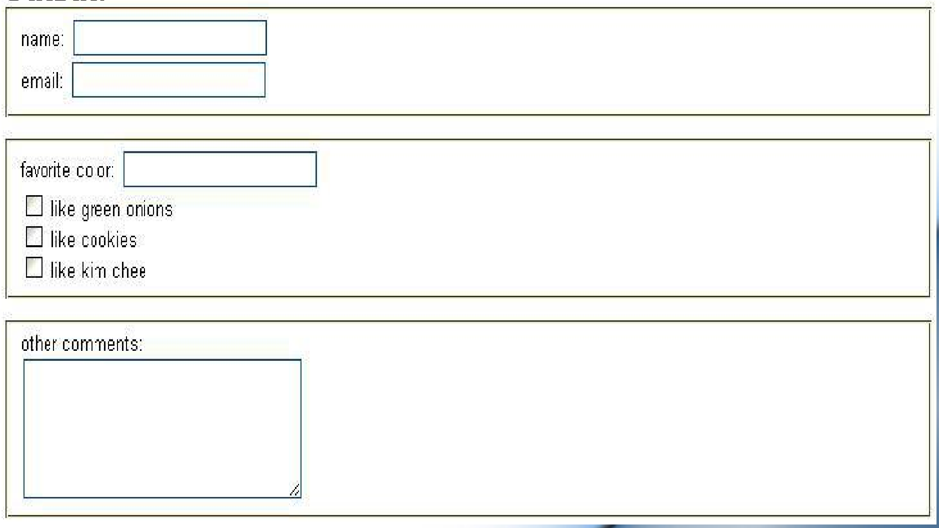
<FIELDSET> defines a group of form elements as being logically related. The browser draws a box around the set of fields to indicate that they are related. For example, a form might contain a few fields about name and email, some fields asking for opinions, and a field for "other comments". <FIELDSET> could be used to group those fields like this:

```
<FIELDSET>
name: <INPUT NAME="realname"><BR>
email: <INPUT NAME="email">
</FIELDSET><P>

<FIELDSET>
favorite color: <INPUT NAME="favecolor"><BR>
<INPUT TYPE=CHECKBOX NAME="onions"> like green
onions<BR>
<INPUT TYPE=CHECKBOX NAME="cookies"> like
cookies<BR>
<INPUT TYPE=CHECKBOX NAME="kimchee"> like kim
chee<BR>
</FIELDSET><P>

<FIELDSET>
other comments:<BR>
<TEXTAREA NAME="comments" ROWS=5
COLS=25></TEXTAREA>
</FIELDSET>
```

Output:



<LEGEND>:

<LEGEND> is used with <FIELDSET> to give a title to each set of fields. Attribute of <LEGEND> is ALIGN. ALIGN align the <LEGEND> text left, center, or right

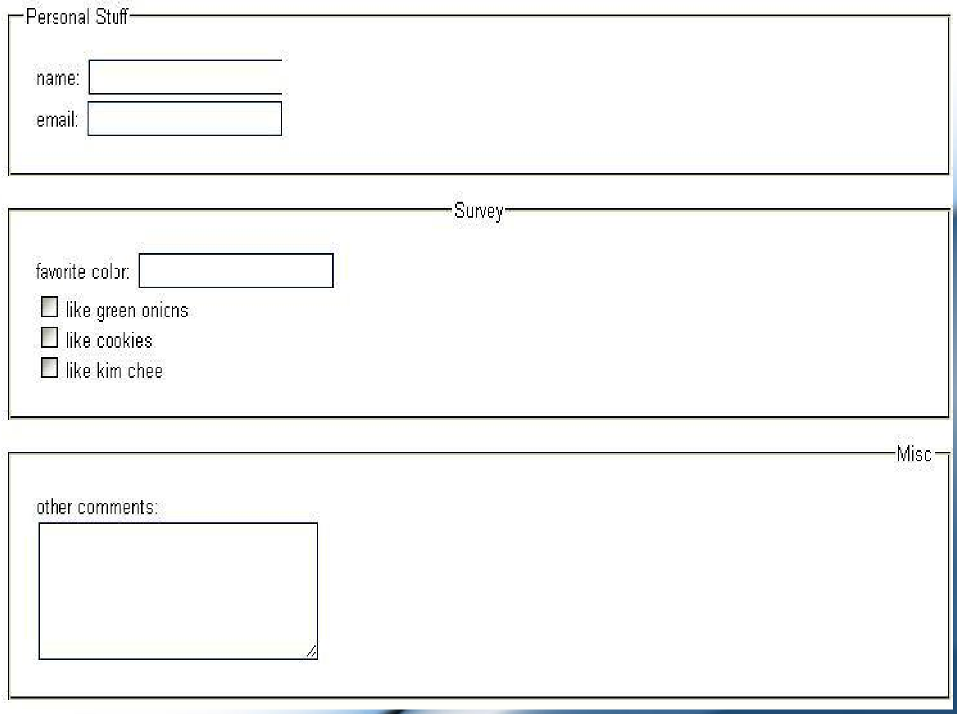
```
<FIELDSET>
<LEGEND ALIGN=LEFT>Personal Stuff</LEGEND><P>
name: <INPUT NAME="realname"><BR>
```

```
email: <INPUT NAME="email">
</FIELDSET><P>

<FIELDSET>
<LEGEND ALIGN=CENTER>Survey</LEGEND><P>
favorite color: <INPUT NAME="favecolor"><BR>
<INPUT TYPE=CHECKBOX NAME="favecolor"> like green
onions<BR>
<INPUT TYPE=CHECKBOX NAME="onions">     like
cookies<BR>
<INPUT TYPE=CHECKBOX NAME="kimchee">   like kim
chee<BR>
</FIELDSET><P>

<FIELDSET>
<LEGEND ALIGN=RIGHT>Misc</LEGEND><P>
other comments:<BR>
<TEXTAREA NAME="comments" ROWS=5
COLS=25></TEXTAREA>
</FIELDSET>
```

Output:



6.18 FORM FIELD EVENT HANDLERS

Following are the event handlers which can be used with form fields:

onClick:

onClick gives the script to run when the user clicks on the input. onClick applies to buttons (submit, reset, and button),checkboxes, radio buttons, and form upload buttons. onClick is mostly used with plain button type inputs:

```
<FORM>
<TABLE BORDER CELLPADDING=3>
<TR>
<TD><NOBR>radius: <INPUT NAME="Circle_radius"
SIZE=4></NOBR></TD>
<TD><INPUT TYPE=BUTTON OnClick="Circle_calc(this.form);"
VALUE="calculate"></TD>
<TD ALIGN=RIGHT BGCOLOR="#AACCFF">
<NOBR>circumference: <INPUT NAME="Circle_circumference"
SIZE=9></NOBR><BR>
<NOBR>area: <INPUT NAME="Circle_area" SIZE=9></NOBR></TD>
</TR>
</TABLE>
</FORM>
```

onClick can return a value to possibly cancel the action the button would normally perform. For example, we can use onClick to double check if the user really wants to reset the form data. In this form, if you click on the reset button, you get a dialog box asking if you really want to cancel. If you choose "cancel", the form is not reset.

```
<INPUT TYPE=RESET onClick="return confirm('Are you sure you
want to reset the form?')" >
```

OnChange:

onChange specifies script code to run when the data in the input field changes. onChange applies to input fields which accept text, namely text and password fields. (<TEXTAREA> and <SELECT> fields also use onChange.)

The onChange event is triggered when the contents of the field changes. For example, when the user enters an email address in this form, a script does some basic validity checking on the value entered:

```
<SCRIPT TYPE="text/javascript">
<!--
function checkEmail(email)
{
```

```
if(email.length > 0)
  {
  if (email.indexOf(' ') >= 0)
    alert("email addresses cannot have spaces in them");
  else if (email.indexOf('@') == -1)
    alert("a valid email address must have an @ in it");
  }
}
//-->
</SCRIPT>
<FORM ACTION="../cgi-bin/mycgi.pl" METHOD=POST>
name:       <INPUT NAME="realname"><BR>
email:      <INPUT NAME="email"
onChange="checkEmail(this.value)"><BR>
destination: <INPUT NAME="desination">
</FORM>
```

Because onChange only occurs when the value changes, the user is only warned once about a bad value. That's generally the most desirable way to do error checking. Even if the value is wrong, most users prefer to be warned just once.

onFocus and onBlur:

onFocus is the event handler for when the input field gets the focus. onBlur is the event handler for when the input field loses the focus. The "focus" indicates which object in the window reacts to keyboard input. If a text field has the focus, then if you type something in the keyboard, the letters appear in that field. Focus shifts from one object to another usually by clicking on them with the mouse or by hitting the tab key.

An example of onFocus and onBlur used to change the status bar. Notice that we use onFocus to give the message we want, and onBlur to change back to the default.

```
<FORM ACTION="../cgi-bin/mycgi.pl" METHOD=POST>
name: <INPUT
      NAME="realname"
      onFocus = "window.status='Enter your name'"
      onBlur  = "window.status=window.defaultStatus"
      ><BR>
email: <INPUT
      NAME="email"
      onFocus = "window.status='Enter your email address'"
```

```
    onBlur  = "window.status=window.defaultStatus"
    >
<P><INPUT TYPE=SUBMIT VALUE="submit">
</FORM>
```

onKeyPress:

```
<HTML>
<HEAD>
<SCRIPT TYPE="text/javascript">
<!--
function numbersonly(myfield, e, dec)
{
var key;
var keychar;
if (window.event)
  key = window.event.keyCode;
else if (e)
  key = e.which;
else
  return true;
keychar = String.fromCharCode(key);
// control keys
if ((key==null) || (key==0) || (key==8) ||
  (key==9) || (key==13) || (key==27) )
  return true;
// numbers
else if ((("0123456789").indexOf(keychar) > -1))
  return true;
// decimal point jump
else if (dec && (keychar == "."))
  {
  myfield.form.elements[dec].focus();
  return false;
  }
else
  return false;
}
//-->
</SCRIPT>
</HEAD>
<BODY>
```

```
<FORM ACTION="../cgi-bin/mycgi.pl" METHOD=POST>
U.S. ZIP Code:
  <INPUT NAME="dollar" SIZE=5 MAXLENGTH=5
   onKeyPress="return numbersonly(this, event)">
  <INPUT TYPE=SUBMIT VALUE="go">
</FORM>
</BODY>
</HTML>
```

In the above code whenever a user enters any value through keyboard (i.e. typing any key), onKeyPress event handler is called. When this event handler is triggered function numbersonly() is called which allows only numbers to be entered.

19. PASSING FORM DATA

Whenever we are creating HTML forms, our objective is to retrieve the values entered by the user. These values are called form data. This form data can be passed to another page using method and action attributes (discussed in <FORM> tag) of form tag. The page name specified in form’s action attribute is the one to which form data is passed, and then processed by this page.

```
<HTML>
<BODY>
<FORM ACTION="insert.php" METHOD="post">
Firstname: <INPUT TYPE="text" NAME="firstname" />
Lastname: <INPUT TYPE="text" NAME="lastname" />
Age: <INPUT TYPE="text" NAME="age" />
<INPUT TYPE="submit" />
</FORM>
</BODY>
</HTML>
```

In the above form user will enter first name and last name in text boxes. When form is submitted insert.php page is called and form data (i.e. first name and last name) is passed to this page. Passing data from insert.html to insert.php is done through post method. Once the form data is with insert.php, it can process the data.

20. SUMMARY

In this chapter we discussed role of form tag, different elements (input, select, textarea, label, option, etc.) and their attributes that can go within form tags. We also discussed how to use event handlers in form.

Answers to check your progress

1.b	2. d	3. a	4.b	5.c
6.a	7.a	8.c		

6.21 EXERCISE

6.21.1 Questions

1. Explain <FORM> tag and its attributes
2. Explain text field, password field, hidden field form elements and how they can be placed on web page.
3. How to put Drop down box in a web page?
4. Explain <fieldset> and <legend> tags in detail.
5. Explain with any four event handlers that can be used with form fields.
6. Explain how to place radio buttons and check boxes on web page.

6.21.2 Programs

1. Design a feedback form that will have following fields:

   Faculty’s name (text field), year (text field), Is professor explaining properly: yes, no (radio button), Any Comments (text area), submit button.

2. Design a registration form that will have following fields:

   User name, password, address, phone no., Gender (male, female), emailed, and a submit button. (Use appropriate form elements wherever necessary)

3. Design a form that will have following fields:

   A text field to enter a sting, four radio buttons for colors (red, green, blue, yellow), and a submit button. Whenever user enters a string, selects a color and submits the form the string should appear in that color.

❖❖❖❖

7

JAVASCRIPT

Unit Structure

1. OBJECTIVE

After reading this chapter you will be able to –

- Write small javascrips with operators and functions .
- Use of variables, datatypes, control statements.
- Identify client and server side scripts
- Use javascript objects and their inbuilt properties.

7.1 INTRODUCTION

- Javascript is object based scripting language based on c++. Scripting language is a light weight programming language which easy to learn and understand. It is generally used for small applicatons.
- Javascript was basically designed to add interactivity in HTML pages and is directly embedded into HTML.
- Javascript is free to use by anyone.
- Javascript is interpreted language ie is not precompiled before execution. As javascript is interpreted, it is platform independent.

- Java is a full fledged complex programming language developed by sun microsystem. Javascript is developed by netscape communications and is no sub language of java.
- Javascript is originally a scripting language developed by European Computer Manufacturer’s association (ECMA)
- Javascript that runs at the client side (ie at the client’s browser) is client side java script (CCJS) and javascript that runs at the server is serverside java script (SSJS)
- Javascript being object oriented, uses number of built in javascript as well as objects can be created.
- Every object has properties and methods. Property is value(s) associated with an object. Methods are actions associated with an object.
- Example:
```
<scripttype="text/javascript">
        document.write("This message is written by JavaScript");
    </script>
```
  in above example, ‘document’ is object and write() is a method of document object.
- Javascript runs in a web browser, and when a script written by a third party is executed on the browser, there is a risk of running a spyware or a virus program.
- Hence, each time javascript is loaded on the browser implements a security policy designed to minimise the risk of such unknown code.
- Security policy is set of rules governing what scripts can do under which circumstances.
- Modern javascript security is based upon Java. Scripts downloaded are isolated from the operating system and then executed. This is known as the ‘sandbox’ model. Some scripts are often stored randomly here and there. And hence, many times obtain more power than expected by design or by accident.
- Scripts in general are given limited access and more access is only given with the user consent. Taking a consent for every execution is not a practical solution.
- Scripts from ‘trusted’ source are many times excluded from this consent procedure.
- A policy called ‘same origin’ does not block scripts coming from the same origin as trusted scripts. This same origin check is performed on all methods of windows object, also on embedded and externally linked objects.

7.3 JAVASCRIPT OPERATORS

Operator	Name	Description	Example	Result	Note
=	Assignment	Assigns value to a variable.	y = 5; x = y = 5;	y = 5 x=5 and y=5	Primitive data types get direct values where as functions are pointers to the variables.
+=	Plus equal to		x += y	x =10	Same as operator equal to. That is, x+=y means that x=x+y and so on.
-=	Minus equal to		x -= y	x = 0	
*=	Multiply equal to		x *= y	x = 25	
/=	Divide equal to		x /= y	x =1	
%=	Modulo equal to		x %= y	x = 0	
Arithmetic					
+	Addition	Adds values	x = y + 2;	x=7	
-	Subtraction	Subtracts second from first value	x = y – 2;	x=3	
*	Multiplication	Multiplies two values	x = y * 2;	x=10	
/	Division	Divides	x = y / 2;	x=2.5	
%	Modulus	Divides and gives the remainder	x = y % 2;	x=1	
++	Increment	Increments value by 1	x = ++y;	x=6	
--	Decrement	Decrements value by 1	x = --y;	x=4	
Logical					
&&	And	Logically ands	(x<10 && y>1)	TRUE	As value of x as of now is 4 and that of y is 5
‖	Or	Logically Ors	(x==5 ‖ y==5)	TRUE	As value of x <> 5 but that of y = 5

!	Not	Negation	!(x==y)	TRUE	As x and y are not equal.
String					
+	Concatenation	Connects two strings.	If x = “5” and y= “5” then	x+y=“55”	
Conditional					
?	Variable =(condition)?value1:value2		If a = 0, b = 7 a=(b= =10)?5:8	a = 8, b = 7	If condition is satisfied, first value is assigned, else the second value is assigned.
Short circuit evaluation					
&&	Sand	Short circuit and	If x = 0; p = 66 x=p && p.getvalue()	x = 66	If p is not null, then assign value of p to x.
‖	SOr	Short circuit or	If x=0, default=5 x=default‖10	x=5	If there is a default, assign default; else assign the given value.
,	Comma		x=4, y= 5;	x=4 y=5	Executes all expressions from left to right.
Delete		Used in functions	Delete <object>	Object undefined	Deletes the properties from objects and array elements from arrays making them undefined.

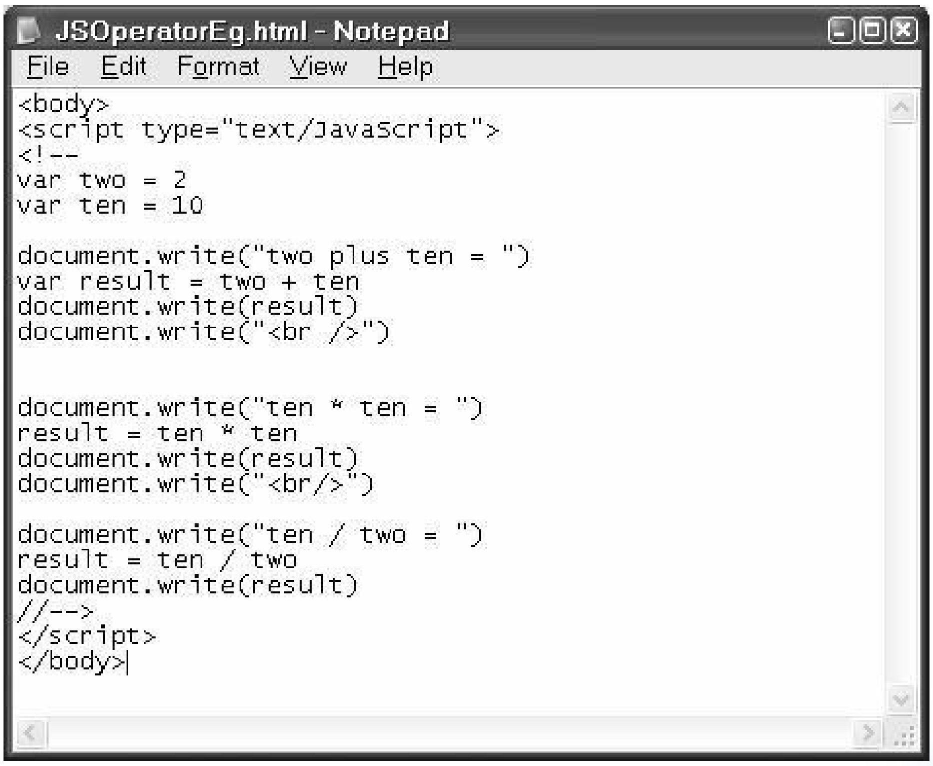
This	Can be used in following contexts –
	• As global
	• Function context
	Simple call
	As object method
	Prototype chain
	• As a constructor
	• As getter or setter.
			this.window
			return this;
			return this.value		

New		Creates new instance of the object	var x= new Fn()	x becomes new instance of function Fn and gets all its properties and methods.	

Void		Evaluates expression and returns undefined	Void expr	Undefined.	

Javascript Operators Examples

1.



JSOperatorEg.html - Notepad

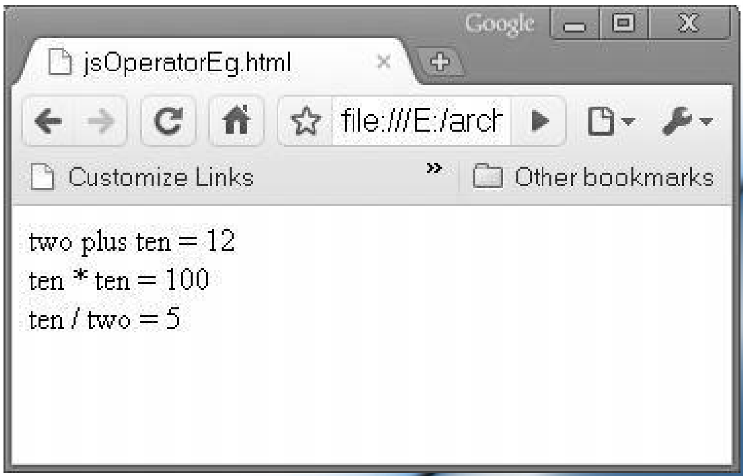
```
<body>
<script type="text/JavaScript">
<!--
var two = 2
var ten = 10

document.write("two plus ten = ")
var result = two + ten
document.write(result)
document.write("<br />")


document.write("ten * ten = ")
result = ten * ten
document.write(result)
document.write("<br/>")

document.write("ten / two = ")
result = ten / two
document.write(result)
//-->
</script>
</body>
```

Output



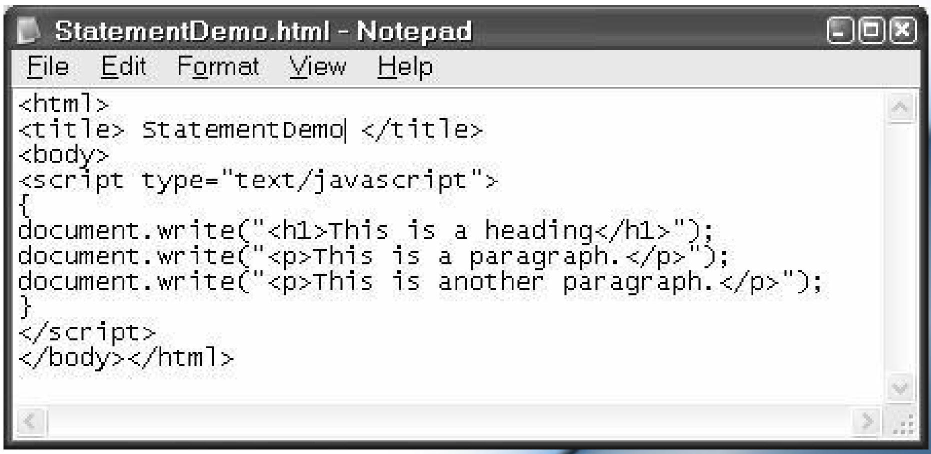
jsOperatorEg.html

file:///E:/arch

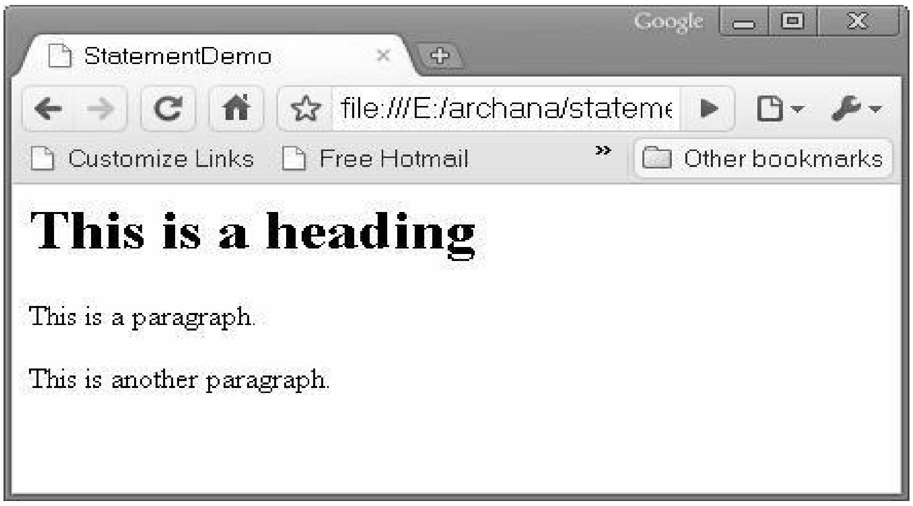
```
two plus ten = 12
ten * ten = 100
ten / two = 5
```

7.4 JAVASCRIPT STATEMENTS

- JavaScript is a sequence of statements to be executed by the browser. Browser executes the statements in the same order as they are written.
- JavaScript is case sensitive with all syntax, variable and function names.
- The semicolon at the end of line is optional (according to the JavaScript standard), and the browser is supposed to interpret the end of the line as the end of the statement. However, semicolon at the end of line is good programming practice. Also, it enables us to write multiple statements on the same line.
- JavaScript statements can be grouped together in blocks. Blocks start with a left curly bracket {, and end with a right curly bracket }. The purpose of a block is to make the sequence of statements execute together.
- A block is normally used to group the statements in a function or condition.
- A general example of block is –



```
<html>
<title> StatementDemo </title>
<body>
<script type="text/javascript">
{
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph.</p>");
}
</script>
</body></html>
```



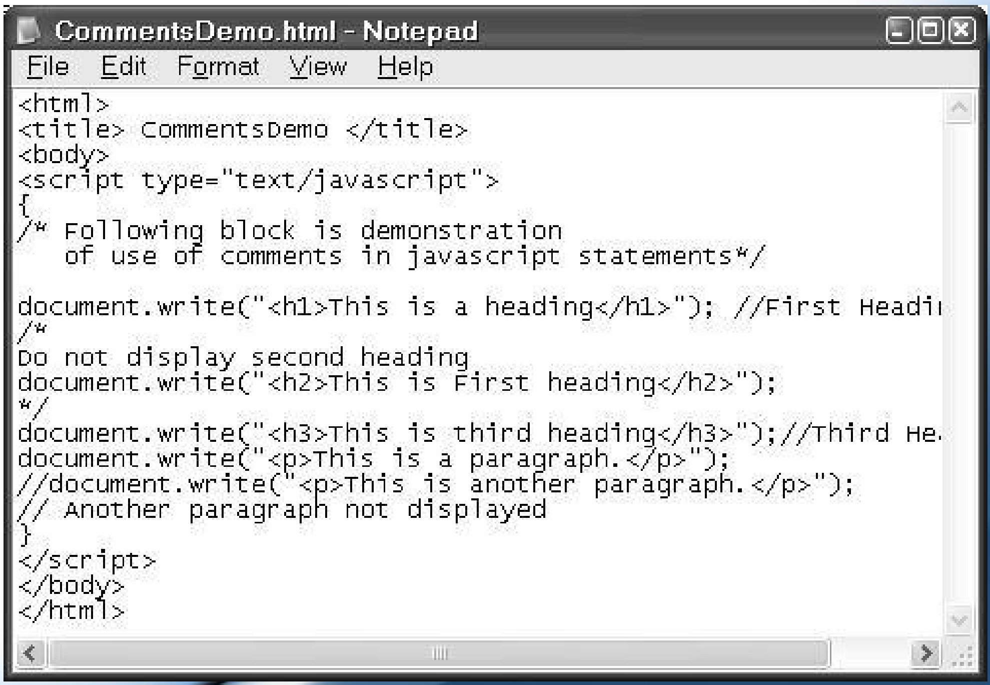
This is a heading

This is a paragraph.

This is another paragraph.

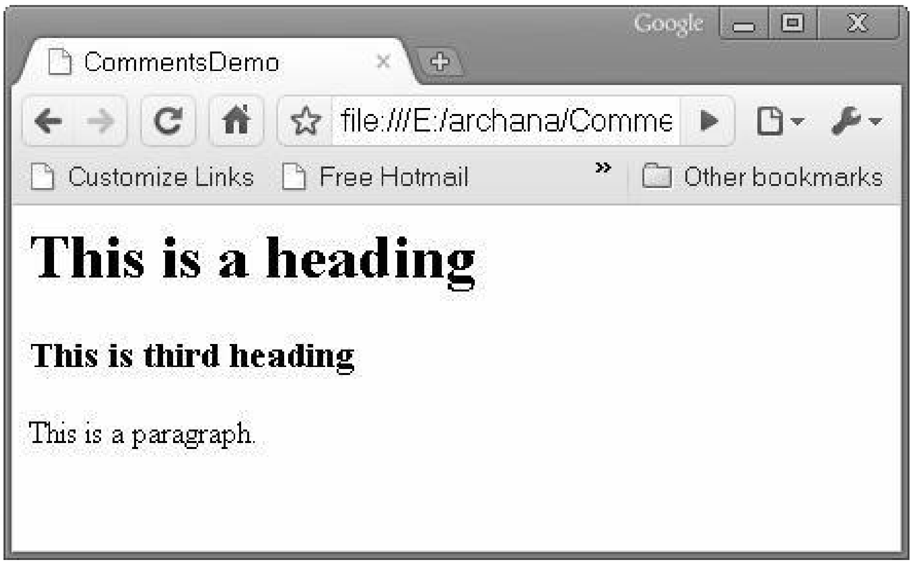
JavaScript Comments

- JavaScript comments can be used to make the code more readable.
- Comments can be added to explain the JavaScript.
- Comments can be added at end of a line.
- Single line comments start with //.
- Multi line comments start with /* and end with */.
- The comment is used to prevent the execution of a single code line or a code block. This can be suitable for debugging
- Following example demonstrates use of comments in javascript code.



```
<html>
<title> CommentsDemo </title>
<body>
<script type="text/javascript">
{
/* Following block is demonstration
   of use of comments in javascript statements*/

document.write("<h1>This is a heading</h1>"); //First Headi
/*
Do not display second heading
document.write("<h2>This is First heading</h2>");
*/
document.write("<h3>This is third heading</h3>");//Third He
document.write("<p>This is a paragraph.</p>");
//document.write("<p>This is another paragraph.</p>");
// Another paragraph not displayed
}
</script>
</body>
</html>
```



This is a heading

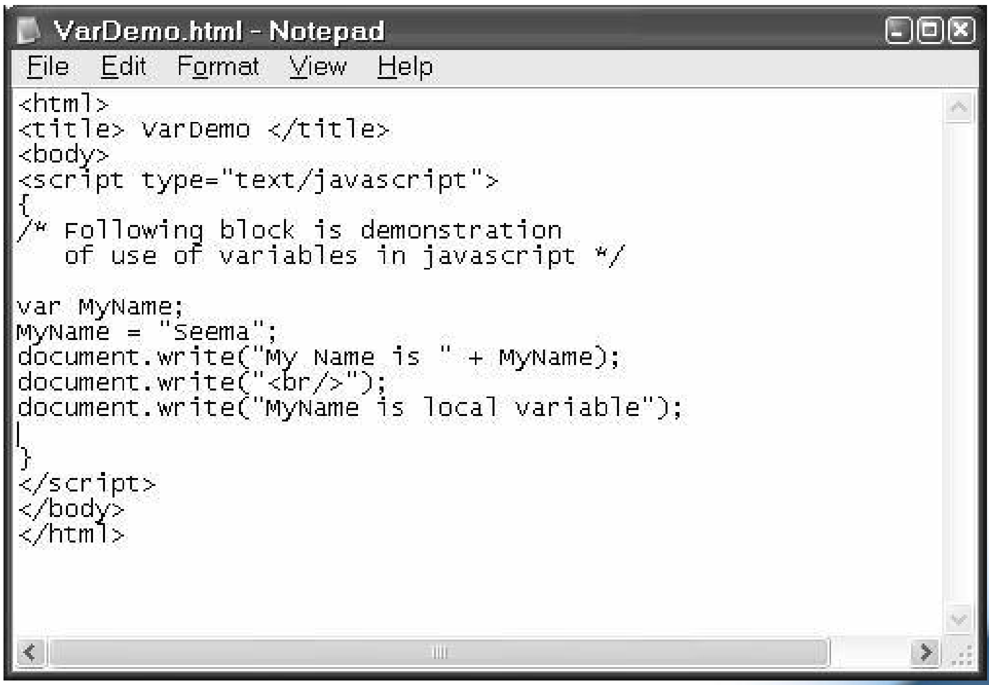
This is third heading

This is a paragraph.

Javascript Variables (Var statement)

- Variables are "containers" for storing information. This information can be values or expressions.
- A variable can have a short name, like x, or a more descriptive name, like MyName.
- Rules for JavaScript variable names:
    - Variable names are case sensitive (y and Y are two different variables)
    - Variable names must begin with a letter or the underscore character
- Creating variables in JavaScript is most often referred to as "declaring" variables.
- You declare JavaScript variables with the var keyword like var x;
- After the declaration shown above, the variables are empty (they have no values yet). However, you can also assign values to the variables when you declare them like var x = 10; After the execution of this statement, the variable x will hold the value 10
- A variable declared within a JavaScript function becomes **LOCAL** and can only be accessed within that function. (the variable has local scope).
- You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.
- Local variables are destroyed when you exit the function.
- Variables declared outside a function become **GLOBAL**, and all scripts and functions on the web page can access it.

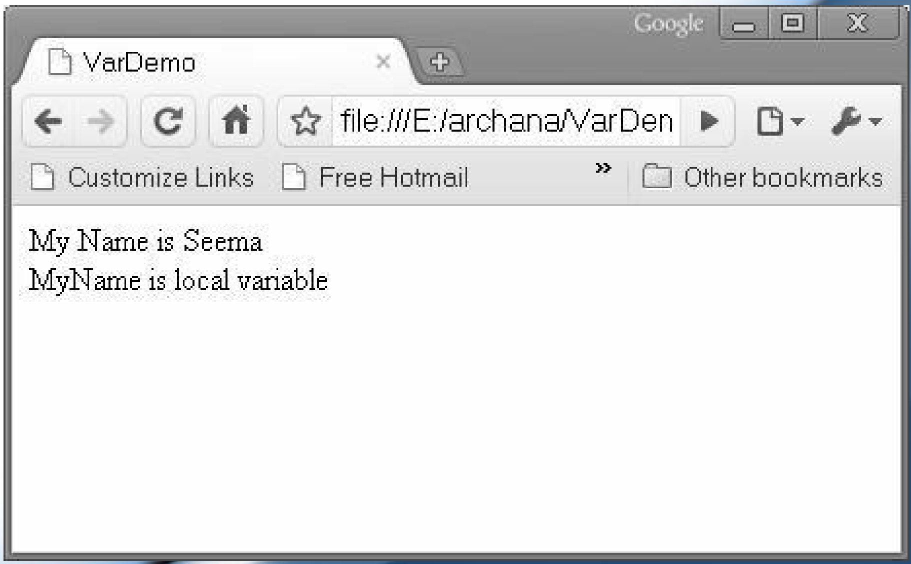
- Global variables are destroyed when you close the page.
- If you declare a variable, without using "var", the variable always becomes GLOBAL.
- If you assign values to variables that have not yet been declared, the variables will automatically be declared as global variables.
- All javascript operators can be used with variables
- Example –



```
<html>
<title> VarDemo </title>
<body>
<script type="text/javascript">
{
/* Following block is demonstration
   of use of variables in javascript */

var MyName;
MyName = "Seema";
document.write("My Name is " + MyName);
document.write("<br/>");
document.write("MyName is local variable");

}
</script>
</body>
</html>
```



```
My Name is Seema
MyName is local variable
```

Conditional statements

- Conditional statements are used to perform different actions based on different conditions.

- In JavaScript we have the following conditional statements:

- **If statement** - This statement is used to execute some code only if a specified condition is true.
    - Syntax:

```
  if(condition)
 {
  code to be executed if condition is
true
    }
```

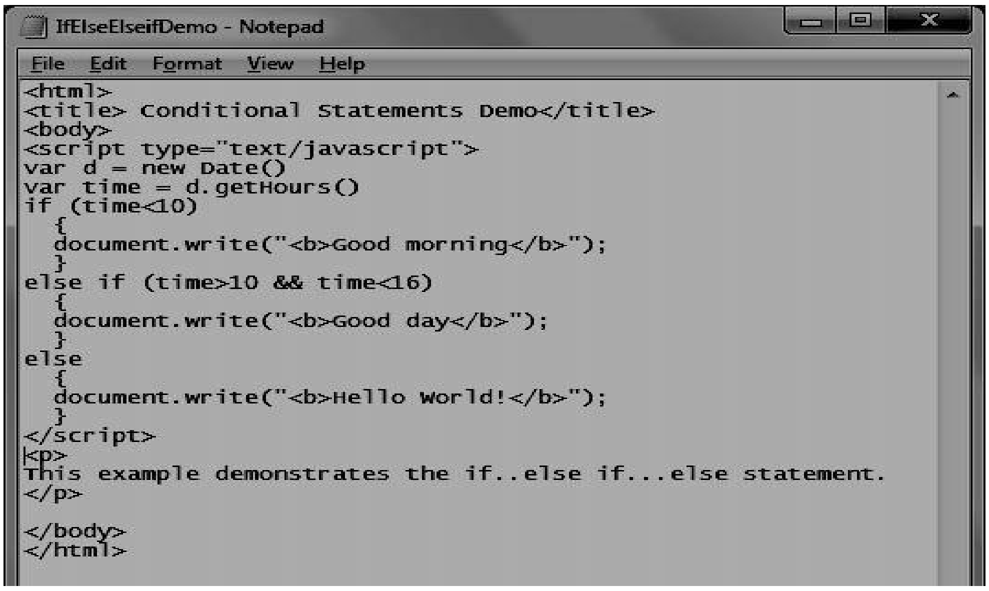
- **If...else statement -** This statement is used to execute some code if the condition is true and another code if the condition is false
    - Syntax:

```
if (condition)
  {
code to be executed if condition is
true
  }
else
  {
  code to be executed if condition is
not true
  }
```

- **If...else if....else statement -** This statement is used to select one of many blocks of code to be executed
    - Syntax:

```
if (condition1)
  {
  code to be executed if condition1 is
true
  }
else if (condition2)
  {
  code to be executed if condition2 is
true
  }
else
  {
  code to be executed if neither
condition1 nor condition2 is true
  }
```

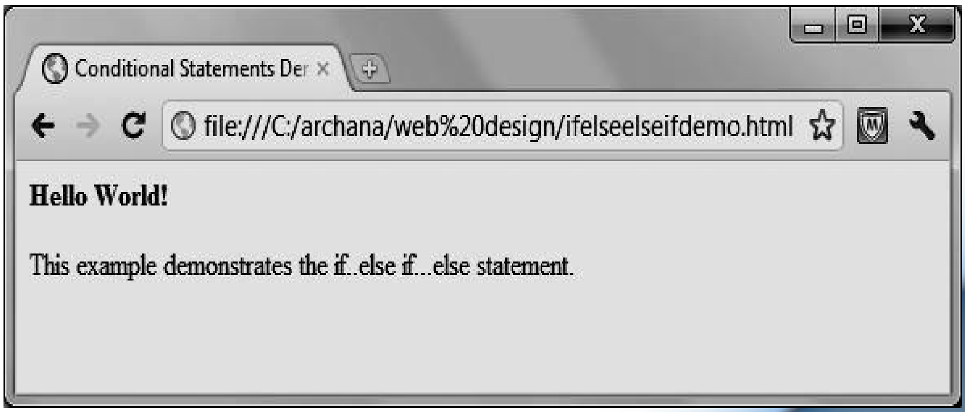
Example



```
<html>
<title> Conditional Statements Demo</title>
<body>
<script type="text/javascript">
var d = new Date()
var time = d.getHours()
if  (time<10)
  {
  document.write("<b>Good morning</b>");
  }
else if (time>10 && time<16)
  {
  document.write("<b>Good day</b>");
  }
else
  {
  document.write("<b>Hello World!</b>");
  }
</script>
<p>
This example demonstrates the if..else if...else statement.
</p>

</body>
</html>
```

Output


Conditional Statements Der ×
file:///C:/archana/web%20design/ifelseelseifdemo.html

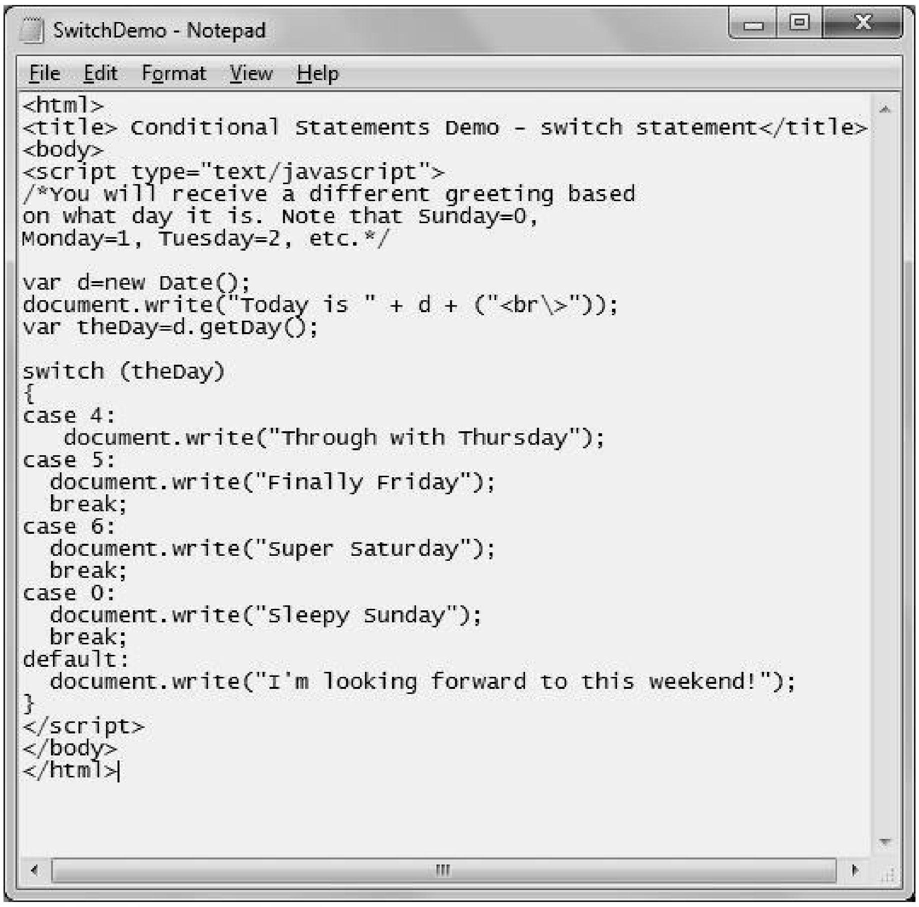
Hello World!

This example demonstrates the if..else if...else statement.

- **switch statement -**This statement is another way to select one of many blocks of code to be executed.
    - Syntax -

```
switch(n)
{
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  code to be executed if n is different
from case 1 and 2
}
```

Example:

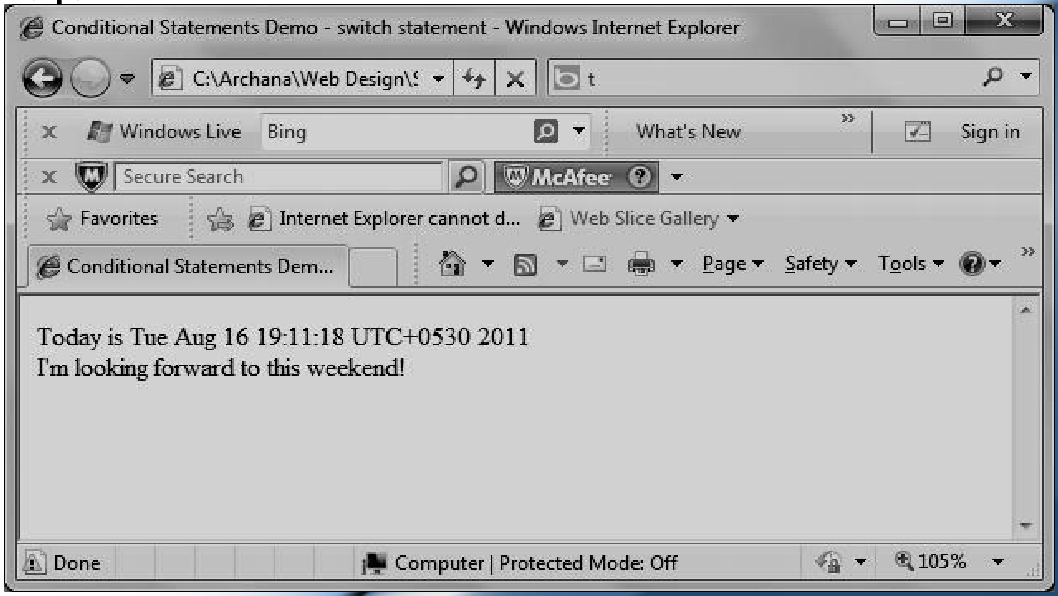


```
<html>
<title> Conditional Statements Demo - switch statement</title>
<body>
<script type="text/javascript">
/*You will receive a different greeting based
on what day it is. Note that Sunday=0,
Monday=1, Tuesday=2, etc.*/

var d=new Date();
document.write("Today is " + d + ("<br\>"));
var theDay=d.getDay();

switch (theDay)
{
case 4:
   document.write("Through with Thursday");
case 5:
  document.write("Finally Friday");
  break;
case 6:
  document.write("Super Saturday");
  break;
case 0:
  document.write("Sleepy Sunday");
  break;
default:
  document.write("I'm looking forward to this weekend!");
}
</script>
</body>
</html>
```

Output


Conditional Statements Demo - switch statement - Windows Internet Explorer
C:\Archana\Web Design\
Windows Live
Bing
What's New
Sign in
Secure Search
McAfee
Favorites
Internet Explorer cannot d...
Web Slice Gallery
Conditional Statements Dem...
Page
Safety
Tools
Done
Computer | Protected Mode: Off
105%

```
Today is Tue Aug 16 19:11:18 UTC+0530 2011
I'm looking forward to this weekend!
```

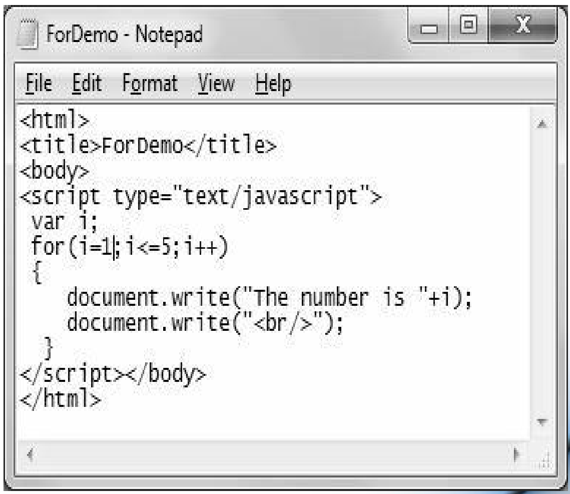
Loop statements

- Loops execute a block of code a specified number of times, or while a specified condition is true.
- Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.
- In JavaScript, there are two different kind of loops:
    - for - loops through a block of code a specified number of times. It can be used only when it is known in advance, how many times we have to run the loop.
    - while - loops through a block of code while a specified condition is true.
- For Loop Syntax:

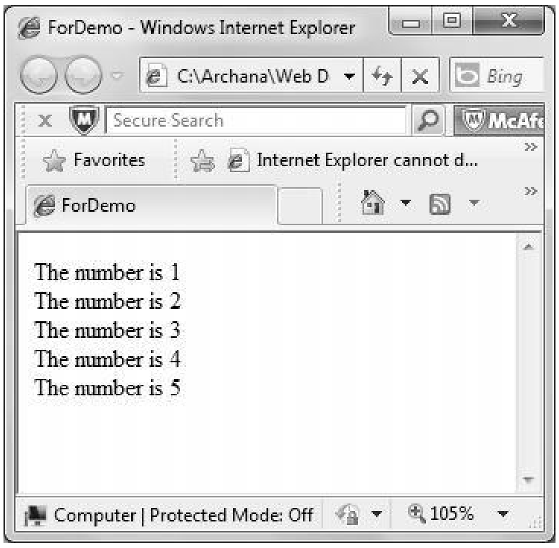
```
for
(variable=startvalue;variable<=endvalue;variabl
e=variable+increment)
{
code to be executed
}
```

- **Example:**

Javascript given below will print 5 numbers. Each time, value of the variable is incremented by 1.



```
<html>
<title>ForDemo</title>
<body>
<script type="text/javascript">
 var i;
 for(i=1;i<=5;i++)
 {
    document.write("The number is "+i);
    document.write("<br/>");
  }
</script></body>
</html>
```


ForDemo - Windows Internet Explorer
C:\Archana\Web D
Bing
Secure Search
McAfee
Favorites
Internet Explorer cannot d...
ForDemo
Computer | Protected Mode: Off
105%

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

- While loop syntax

```
while (var<=endvalue)
  {
  code to be executed
  }
```

- While loop can be used with any comparison operator.
- **Do While** loop is a variation to the while loop. In this case block will be executed at least once, as the statements are executed before the condition is tested. Syntax for do while is as follows –

```
do
  {
  code to be executed
  }
while (var<=endvalue);
```

- Consider the example where number is printed after incrementing it by 1. This is performed while the number is less than or equal to 5. Script and the outputs with while and do while loop are as given below –

```
<html>
<body>
<script type="text/javascript">
var i=6;
do
  {
  document.write("The number is " + i);
  document.write("<br />");
  i++;
  }
while (i<=5);
```

Thank You